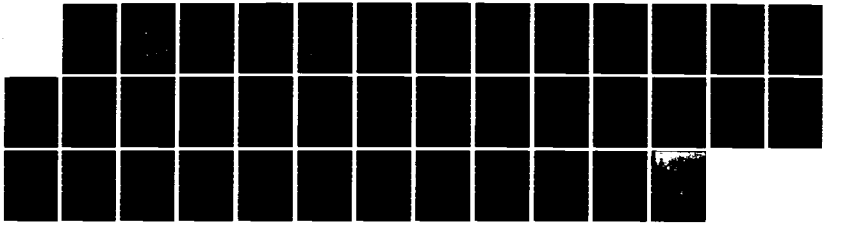
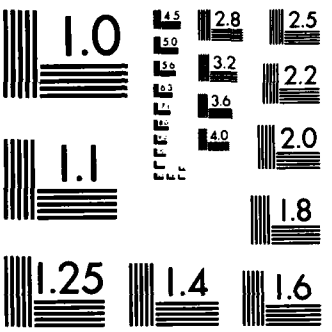
AD-A132 400 ALGORITHMS FOR SINGLE-SIGNAL AND MULTISIGNAL MINIMUM-CROSS-ENTROPY SPECTRUM ANALYSIS(U) NAVAL RESEARCH LAB WASHINGTON DC R W JOHNSON 03 AUG 83 1/1

UNCLASSIFIED NRL-8667 SBI-AD-E000 542 F/G 17/2.1 NL





(2)

NRL Report 8667

Algorithms for Single-Signal and Multisignal Minimum-Cross-Entropy Spectrum Analysis

RODNEY W. JOHNSON

Computer Science and Systems Branch
Information Technology Division

August 3, 1983

NAVAL RESEARCH LABORATORY
Washington, D.C.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: NRL Report 8667	2. GOVT ACCESSION NO.: AD-A132 400
3. RECIPIENT'S CATALOG NUMBER:	
4. TITLE (and Subtitle): ALGORITHMS FOR SINGLE-SIGNAL AND MULTISIGNAL MINIMUM-CROSS-ENTROPY SPECTRUM ANALYSIS	5. TYPE OF REPORT & PERIOD COVERED: Interim report on a continuing NRL problem
	6. PERFORMING ORG. REPORT NUMBER: 7590-290:RJ:pt
7. AUTHOR(s): Rodney W. Johnson	8. CONTRACT OR GRANT NUMBER(s):
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Naval Research Laboratory, Washington, DC 20375	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 61153N; RR0210542; NRL Problem 75-0107-0-3
11. CONTROLLING OFFICE NAME AND ADDRESS: Office of Naval Research, Arlington, VA 22217	12. REPORT DATE: August 3, 1983
	13. NUMBER OF PAGES: 36
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office):	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE:

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Spectral analysis	Information theory
Maximum entropy	Discrimination information
Minimum cross entropy	Itakura-Saito distortion
Directed divergence	Noise suppression

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Two algorithms are presented that implement multisignal minimum-cross-entropy spectral analysis (MCESA), a method for estimating the power spectrum of one or more independent signals when a prior estimate for each is available and new information is obtained in the form of values of the autocorrelation function of their sum. Single-signal MCESA is included as a special case. One of the algorithms is slow, but general: the prior spectrum estimates and the resulting (posterior) spectrum estimates are represented by discrete-frequency approximations

(Continued)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20 ABSTRACT (Continued)

with arbitrarily spaced frequencies, and the autocorrelation values may be given at arbitrarily spaced lags. The other algorithm is considerably faster and applies to an important special case: the prior and posterior spectrum estimates are of the all-pole form that results from maximum-entropy (or linear-predictive) spectral analysis, and the autocorrelation values are given at equispaced lags beginning at zero.

CONTENTS

ALGORITHMS FOR SINGLE-SIGNAL AND MULTISIGNAL MINIMUM-CROSS-ENTROPY SPECTRUM ANALYSIS

INTRODUCTION

Multisignal minimum-cross-entropy spectral analysis (multisignal MCESA) is a method for estimating the power spectrum of one or more independent signals when a prior estimate for each is available and new information is obtained in the form of values of the autocorrelation function of their sum [1]. This method is a generalization of MCESA [2] and reduces to it when there is only one signal. One application of multisignal MCESA is to noise reduction [3]. If one obtains autocorrelation measurements for a signal with independent additive interference, and if one has some prior knowledge, expressible as spectral estimates, concerning the signal and the noise, the method yields new signal- and noise-spectrum estimates that take both the prior estimates and the autocorrelation information into account.

We here present two algorithms that implement multisignal MCESA. One is slow, but general; the other is considerably faster and applies to an important special case.

The first algorithm accepts as inputs prior estimates of the power spectra of one or several independent signals and measured autocorrelation values for the sum of the signals, and it produces as outputs posterior spectrum estimates that are consistent with the autocorrelation estimates. The spectra are represented by discrete-frequency approximations—lists of spectral powers at specified frequencies. The algorithm implements the methods of Refs. 1 and 2 in full generality: the discrete frequencies may be arbitrarily spaced, and the autocorrelations may be given at arbitrarily spaced lags. The algorithm is iterative; at each iteration, it computes frequency-domain quantities (spectra) and time-domain quantities (autocorrelations and Lagrange multipliers) that are related to each other by procedures equivalent in complexity to the naive (slow) algorithm for the discrete Fourier transform.

The second algorithm applies to the special case in which the prior spectral estimates are of the all-pole form that results from maximum-entropy spectral analysis (MESA) [4,5] or linear-predictive coding (LPC) [6]. Such spectra may be represented in various ways by finite families of parameters that determine them uniquely—for example by a gain and inverse filter coefficients or by a gain and reflection coefficients. In particular, the spectra may be represented by a finite number of autocorrelation values. This algorithm accepts prior autocorrelation estimates for the individual signals in place of prior spectral estimates. The prior autocorrelation estimates for the individual signals and the measured autocorrelations of the sum of the signals are given at equispaced lags beginning at zero. The posterior spectral estimates that result from data of this form are all-pole spectra like the prior estimates, though not necessarily of the same order. The outputs from the algorithm may be posterior autocorrelation estimates or, alternatively, any of various equivalent families of LPC parameters.

The next section gives enough background for a statement of the problem to be solved; for a fuller discussion, see Refs. 1 and 2. The third section presents the first algorithm; the fourth section presents the second algorithm. The fifth section contains a general discussion.

Manuscript approved November 2, 1982.

THE PROBLEM

Consider a number p of unknown real, independent, band-limited, discrete-spectrum signals with no dc component; let their spectra be $\boldsymbol{S}_h$, $h = 1, \ldots, p$, with spectral powers S_{hk} at frequencies $\pm f_k$, where $f_k > 0$ for $k = 1, \ldots, n$. Let P_{hk} be a *prior* estimate for S_{hk}. Now suppose we are given values R_r^{tot} of the autocorrelation function of the sum of the signals for lags t_r, $r = 0, \ldots, m$. Under the assumption that the processes are independent, we write R_r^{tot} as a sum of individual autocorrelations,

$$R_r^{\text{tot}} = \sum_{h=1}^{p} R_{hr},$$

where

$$R_{hr} = \sum_{k=1}^{n} 2S_{hk} \cos 2\pi f_k t_r.$$

We wish to use the autocorrelation information to obtain improved, *posterior* estimates $\boldsymbol{Q}_h$ of the $\boldsymbol{S}_h$. The multisignal minimum-cross-entropy estimate, given by Eq. (18) of Ref. 1, is

$$Q_{hk} = Q_{hk}(\boldsymbol{\beta}) = Q_{hk}(\beta_1, \ldots, \beta_m) \tag{1}$$
$$= \frac{1}{\dfrac{1}{P_{hk}} + \displaystyle\sum_{r=0}^{m} \beta_r C_{rk}},$$

where

$$C_{rk} = 2 \cos 2\pi f_k t_r. \tag{2}$$

The parameters β_r, $r = 1, \ldots, m$, are Lagrange multipliers associated with a constrained minimization problem: the cross entropy of a pair of probability distributions related to the $\boldsymbol{P}_h$ and the $\boldsymbol{Q}_h$ is minimized subject to *constraints*

$$R_r^{\text{tot}} = \sum_{h=1}^{p} \sum_{k=1}^{n} Q_{hk}(\boldsymbol{\beta}) C_{rk} \tag{3}$$

[*cf.* Ref. 1, Eq. (21)] that require the posterior estimates $\boldsymbol{Q}_h$ to be consistent with the given values of the summed autocorrelations. The β_r are to be chosen so that Eq. (3) is satisfied. The first algorithm applies in this case, accepting the quantities t_r, R_r^{tot}, f_r, and P_{hk} as inputs and computing the Q_{hk} as outputs.

In the case to which the second algorithm applies, we take the spacing between autocorrelation lags as a unit of time and its reciprocal as a unit of frequency. We thus write $t_r = r$, $r = 0, \ldots, m$. In place of a discrete-frequency approximation, we write spectra as functions of a continuous variable f in the interval $-\frac{1}{2} \leqslant f \leqslant \frac{1}{2}$. Thus we have prior estimates $P_h(f)$ of the spectral power densities $S_h(f)$. Corresponding to Eq. (1) we have

$$Q_h(f) = \frac{1}{\dfrac{1}{P_h(f)} + 2\displaystyle\sum_{r=0}^{m} \beta_r \cos 2\pi f r}, \tag{4}$$

and the constraints to be satisfied are

$$R_r^{\text{tot}} = 2 \sum_{h=1}^{p} \int_0^{1/2} Q_h(f) \cos 2\pi f r \, df.$$

The second algorithm avoids direct computation with spectra P_h and Q_h—that is, it avoids the use of discrete-frequency approximations to P_h and Q_h and numerical integration over frequency. This is possible since, as we will see, all the spectra that occur are of the all-pole (MESA or LPC) form [5,6]

$$Q_h(f) = \frac{E_h}{\left|\sum_{t=0}^{M} a_{ht} e^{2\pi i f t}\right|^2}, \tag{5}$$

where the a_{ht} are inverse filter coefficients and E_h is a gain. This is equivalent to

$$Q_h(f) = \frac{1}{2\sum_{t=0}^{M} \lambda_{ht} \cos 2\pi f t}, \tag{6}$$

where the Lagrange multipliers λ_{ht} are related to E_h and the a_{ht} by

$$\lambda_{h0} = \frac{1}{2E_h} \sum_{s=0}^{M} a_{hs}^2$$

and

$$\lambda_{ht} = \frac{1}{E_h} \sum_{s=0}^{M-t} a_{hs} a_{h,s+t}, \qquad t = 1, \ldots, M. \tag{7}$$

The prior spectra P_h have the all-pole form by assumption; we express them in the form

$$P_h(f) = \frac{1}{2\sum_{t=0}^{M} \lambda_{ht}^0 \cos 2\pi f t}.$$

We may assume that the prior spectra are of order $M \geqslant m$, for we can extend the sequences $\lambda_{h0}, \lambda_{h1}, \ldots$ with zeros if necessary. To see that the posterior spectra must have the same form, we substitute this expression for P_h in Eq. (4). We find that Q_h is given by Eq. (6) with

$$\lambda_{ht} = \begin{cases} \lambda_{ht}^0 + \beta_t, & 0 \leqslant t \leqslant m, \\ \lambda_{ht}^0, & m < t \leqslant M. \end{cases} \tag{8}$$

The autocorrelations R_{ht} of Q_h are related to the λ_{ht} by

$$\begin{aligned} R_{ht} &= \int_0^{1/2} 2Q_h(f) \cos 2\pi f t \, df \\ &= \int_0^{1/2} \frac{\cos 2\pi f t}{\sum_{u=0}^{M} \lambda_{hu} \cos 2\pi f u} \, df, \end{aligned} \tag{9}$$

and the constraints can be expressed as

$$\sum_h R_{hr} = R_r^{\text{tot}}, \qquad r = 0, \ldots, m. \tag{10}$$

Either family of parameters, $\boldsymbol{R}_h = (R_{h0}, R_{h1}, \ldots, R_{hM})$ or $\boldsymbol{\lambda}_h = (\lambda_{h0}, \lambda_{h1}, \ldots, \lambda_{hM})$, is sufficient to determine Q_h, and there are algorithms to compute either family from the other without numerical integration of the right side of Eq. (9). Thus we can work in terms of $\boldsymbol{R}_h$ and $\boldsymbol{\lambda}_h$ rather than Q_h.

The second algorithm begins with the summed autocorrelations R_r^{tot} and the autocorrelations

$$R_{ht}^0 = 2\int_0^{1/2} P_h(f) \cos 2\pi f t \, dt$$

of the prior spectral estimates P_h. It computes the autocorrelations R_{ht} of the posterior spectral estimates Q_h; the Lagrange multipliers λ_{ht} of Eqs. (6) and (9) are also available as outputs.

THE FIRST ALGORITHM

Derivation of the First Algorithm

When Eq. (1) holds, the right-hand side of Eq. (3) for each r is a function of the Lagrange multipliers:

$$F_r(\boldsymbol{\beta}) = \sum_{h=1}^{p} \sum_{k=1}^{n} \frac{1}{\frac{1}{P_{hk}} + \sum_{s=0}^{m} \beta_s C_{sk}} C_{rk}; \tag{11}$$

we need to solve

$$F_r(\boldsymbol{\beta}) = R_r^{\text{tot}}, \qquad r = 1, \ldots, m, \tag{12}$$

for $\boldsymbol{\beta}$. The Newton-Raphson method for approximating solutions to such systems of equations starts with an initial guess $\boldsymbol{\beta}^{(0)}$ at the solution, iteratively computes approximations $\boldsymbol{\beta}^{(1)}, \boldsymbol{\beta}^{(2)}, \ldots,$ by

$$\boldsymbol{\beta}^{(i)} = \boldsymbol{\beta}^{(i-1)} + \Delta\boldsymbol{\beta}^{(i)}, \tag{13}$$

where $\Delta\boldsymbol{\beta}^{(i)}$ satisfies

$$F_r(\boldsymbol{\beta}^{(i-1)}) + \sum_{s=1}^{m} \frac{\partial F_r(\boldsymbol{\beta}^{(i-1)})}{\partial \beta_s^{(i-1)}} \Delta\beta_s^{(i)} = R_r^{\text{tot}}, \tag{14}$$

and stops when the $\boldsymbol{\beta}^{(i)}$ have satisfied some convergence criterion. In matrix notation, for fixed i the solution $\Delta\boldsymbol{\beta}^{(i)}$ of Eq. (14) is given by

$$\Delta\boldsymbol{\beta}^{(i)} = \mathbf{M}^{-1}\boldsymbol{\nu}, \tag{15}$$

where $\boldsymbol{\nu}$ is the vector with components

$$\nu_r = R_r^{\text{tot}} - F_r(\boldsymbol{\beta}^{(i-1)}), \tag{16}$$

and $\mathbf{M}$ is the matrix with elements

$$M_{rs} = \frac{\partial F_r(\boldsymbol{\beta}^{(i-1)})}{\partial \beta_s^{(i-1)}}.$$

Differentiating Eq. (11), we obtain

$$\begin{aligned} M_{rs} &= \sum_{h=1}^{p} \sum_{k=1}^{n} \frac{-C_{rk}C_{sk}}{\left(\frac{1}{P_{hk}} + \sum_{t=0}^{m} \beta_t^{(i-1)} C_{tk}\right)^2} \\ &= \sum_{k=1}^{n} -C_{rk}C_{sk} \sum_{h=1}^{p} Q_{hk}(\boldsymbol{\beta}^{(i-1)})^2. \end{aligned}$$

It follows that $\mathbf{M}$ is given by

$$\mathbf{M} = -\mathbf{N}\mathbf{N}', \tag{17}$$

where the prime denotes transposition and

$$N_{rk} = C_{rk}\left[\sum_{h=1}^{p} Q_{hk}(\boldsymbol{\beta}^{(i-1)})^2\right]^{1/2}. \tag{18}$$

From Eqs. (11), (1), and (18) we obtain

$$F_r(\boldsymbol{\beta}^{(i-1)}) = \sum_{k=1}^{n} N_{rk} \frac{\sum_h Q_{hk}(\boldsymbol{\beta}^{(i-1)})}{\left[\sum_h Q_{hk}(\boldsymbol{\beta}^{(i-1)})^2\right]^{1/2}}. \tag{19}$$

We may therefore express the vector $\boldsymbol{\nu}$ defined in Eq. (16) by

$$\boldsymbol{\nu} = -\mathbf{N}\boldsymbol{w}, \tag{20}$$

where

$$w_k = \frac{-x_k + \sum_h Q_{hk}(\boldsymbol{\beta}^{(i-1)})}{\left[\sum_h Q_{hk}(\boldsymbol{\beta}^{(i-1)})^2\right]^{1/2}}, \tag{21}$$

and the x_k are any solution of

$$\sum_k N_{rk} \frac{x_k}{\left[\sum_h Q_{hk}(\boldsymbol{\beta}^{(i-1)})^2\right]^{1/2}} = R_r^{\text{tot}}.$$

By Eq. (18), this is equivalent to

$$\sum_k C_{rk} x_k = R_r^{\text{tot}}$$

or, in matrix notation,

$$\mathbf{C}\boldsymbol{x} = \boldsymbol{R}^{\text{tot}}. \tag{22}$$

Substituting Eqs. (17) and (20) in Eq. (15), we obtain

$$\Delta\boldsymbol{\beta}^{(i)} = (\mathbf{N}\mathbf{N}')^{-1}\mathbf{N}\boldsymbol{w}.$$

The quantity $(\mathbf{N}\mathbf{N}')^{-1}\mathbf{N}$ (when $\mathbf{N}\mathbf{N}'$ is nonsingular) equals the Moore-Penrose generalized inverse [7] $\mathbf{N}'^{\dagger}$ of $\mathbf{N}'$. Thus,

$$\Delta\boldsymbol{\beta}^{(i)} = \mathbf{N}'^{\dagger}\boldsymbol{w}. \tag{23}$$

(This algorithm was first coded in the programming language APL. Equation (23) is in a convenient form for translation into APL, since there is an APL primitive function [8] that will yield the generalized inverse of a matrix.) We can express the solution of Eq. (22) in terms of the generalized inverse. We assume that Eq. (3) provides m independent conditions on the $\boldsymbol{Q}_h$; thus the rows of $\mathbf{C}$ are linearly independent. This implies that the generalized inverse is a right inverse; then $\mathbf{C}\mathbf{C}^{\dagger}$ is an identity matrix. It therefore suffices to set

$$\boldsymbol{x} = \mathbf{C}^{\dagger}\boldsymbol{R}^{\text{tot}} \tag{24}$$

for a solution of Eq. (22). The essential ingredients of the algorithm are now at hand. We choose an arbitrary starting value for $\boldsymbol{\beta}^{(0)}$, compute the $\boldsymbol{Q}_h(\boldsymbol{\beta}^{(0)})$ by Eq. (1), and define $\boldsymbol{x}$ by Eq. (24). We then enter a loop in which, on the i^{th} iteration, we first compute $\boldsymbol{\beta}^{(i)}$ by Eqs. (13), (18), (21), and (23) and then compute $\boldsymbol{Q}_h(\boldsymbol{\beta}^{(i)})$ by Eq. (1). The loop terminates when the $\boldsymbol{Q}_h(\boldsymbol{\beta}^{(i)})$ have converged by some appropriate criterion. In practice, some modifications of the procedure are necessary to prevent the Newton-Raphson algorithm from behaving badly in case the arbitrary starting point is not near enough to a solution. These are incorporated in the following step-by-step outline of the algorithm as the adjustment to $\Delta\boldsymbol{\beta}$ in Step 6.

Summary of First Algorithm

In what follows we will use a left arrow to denote assignment to a variable of a new value, possibly computed in terms of the current value. Thus, corresponding to Eq. (13) we write $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} + \Delta\boldsymbol{\beta}$ (without superscripts) in Step 7. The inputs to the first algorithm are three vectors, $\boldsymbol{t}$, $\boldsymbol{R}^{\text{tot}}$, and $\boldsymbol{f}$, and a matrix $\mathbf{P}$, listing respectively the lags, autocorrelations, frequencies, and prior spectral estimates, t_r, R_r^{tot}, f_k, and P_{hk}, $r=1, \ldots, m$; $k=1, \ldots, n$; $h=1, \ldots, p$. The output is a matrix $\mathbf{Q}$ listing the posterior spectral estimates Q_{hk}.

The steps of the algorithm are as follows:

Step 1. Compute the matrix $\mathbf{C}$ by Eq. (2) and compute the vector $\boldsymbol{x}$ by Eq. (24).

Step 2. Assign initial values to variables

$$\boldsymbol{\beta} \leftarrow (0, \ldots, 0)$$

$$\mathbf{Q} \leftarrow \mathbf{P}.$$

Step 3. (Begin main loop.) Save the current value of $\mathbf{Q}$ for later comparison with the new value:

$$\mathbf{Q}^0 \leftarrow \mathbf{Q}.$$

Step 4. Compute a tentative value of $\Delta\boldsymbol{\beta}$ by Eqs. (18), (21), and (23).

Step 5. Compute tentative values T_{hk} for $1/Q_{hk}$ by

$$T_{hk} \leftarrow \frac{1}{P_{hk}} + \sum_r (\beta_r + \Delta\beta_r) c_{rk}$$

[*cf.* Eq. (1)].

Step 6. If $T_{hk} > 0$ for all h and k, proceed to Step 7. Otherwise, adjust $\Delta\boldsymbol{\beta}$ by the replacements

$$\Delta\beta_r \leftarrow \frac{0.9\Delta\beta_r}{1 - \min\limits_{hk} Q_{hk}^0 T_{hk}}$$

and recompute the T_{hk} by Step 5 before going on to Step 7. (The denominator on the right side was chosen to make the recomputed values of T_{hk} nonnegative. The factor 0.9 is included to assure strict positivity. The value 0.9 is rather arbitrary; in principle it could be any positive number less than 1.)

Step 7. Assign new values to variables

$$\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} + \Delta\boldsymbol{\beta}$$

and

$$Q_{hk} \leftarrow 1/T_{hk}.$$

Step 8. If the new value of $\mathbf{Q}$ equals the previous value (saved as $\mathbf{Q}^0$) to within a specified relative tolerance, terminate the algorithm and return $\mathbf{Q}$ as the result. Otherwise go back to Step 3 and repeat from there.

THE SECOND ALGORITHM

In this subsection we sketch a multisignal MCESA algorithm that works in terms of autocorrelations and Lagrange multipliers. The next three subsections discuss useful standard subroutines and derive in detail the major steps of the algorithm. The final subsection presents the algorithm in its entirety.

The algorithm begins with the measured autocorrelations R_r^{tot} of the summed signals and autocorrelations R_{ht}^0 of the prior spectral estimates P_h. It is assumed that the R_r^{tot} are given at equispaced lags $r = 0, 1, \ldots, m$; the P_h are all-pole spectra; and the R_{ht}^0 are given at equispaced lags $t = 0, 1, \ldots, M_h$, where M_h is at least as great as the order of the all-pole spectrum P_h. There is no loss of generality in assuming that R_{ht}^0 is given out to lag $M \geqslant m$ for all h, since there are standard LPC methods for extrapolating autocorrelation sequences—see the following subsection and the subsection "Summary of Second Algorithm." From the families $\boldsymbol{R}_h^0$ of prior autocorrelations R_{ht}^0, one can obtain the corresponding Lagrange multipliers λ_{ht}^0; these satisfy

$$R_{ht}^0 = \int_0^{1/2} \frac{\cos 2\pi fu}{\sum_{u=0}^{M} \lambda_{hu}^0 \cos 2\pi fu}\, df$$

[*cf.* Eq. (9)]. Initial trial values are given to the posterior autocorrelations R_{ht}, the corresponding Lagrange multipliers λ_{ht}, and the parameters β_r. (Initial value assignments $\boldsymbol{R}_h \leftarrow \boldsymbol{R}_h^0$, $\boldsymbol{\lambda}_h \leftarrow \boldsymbol{\lambda}_h^0$, $\boldsymbol{\beta} \leftarrow (0, \ldots, 0)$ are reasonable and appear to work well in practice.) These values are then adjusted iteratively with the aim of simultaneously establishing Eqs. (8), (9), and (10). At each iteration, the $\boldsymbol{\lambda}_h$ are recomputed from the current values of $\boldsymbol{R}_h$; Eq. (9), which determines the functional dependence between $\boldsymbol{R}_h$ and $\boldsymbol{\lambda}_h$, is thus maintained. Adjustments $\Delta\boldsymbol{R}_h$, aimed at establishing Eq. (8), and $\Delta\boldsymbol{\beta}$, aimed at establishing Eq. (10), are computed by a linear approximation involving the partial derivatives $\partial R_{ht}/\partial\lambda_{hu}$. The replacements $\boldsymbol{R}_h \leftarrow \boldsymbol{R}_h + \Delta\boldsymbol{R}_h$, $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} + \Delta\boldsymbol{\beta}$ are made, the $\boldsymbol{\lambda}_h$ are recomputed, and the process is repeated with the new trial values of $\boldsymbol{R}_h$, $\boldsymbol{\lambda}_h$, and $\boldsymbol{\beta}$. The repetition continues until the values of $\boldsymbol{R}_h$ are in sufficiently good agreement on two successive iterations.

Some LPC Algorithms

Certain standard LPC signal-processing procedures can be used as subroutines at several points in the algorithm. FORTRAN implementations of all of them are incorporated in the subroutine LPTRN in Section 4.3 of Ref. 9.

To obtain Lagrange multipliers from autocorrelations, we can start with the Levinson recursion [10]. This yields inverse filter coefficients $a_{h0} = 1, a_{h1}, \ldots, a_{hM}$ and a gain E_h such that

$$R_{ht} = \int_0^{1/2} \frac{2E_h \cos 2\pi ft}{\left|\sum_{s=0}^{M} a_{hs} e^{2\pi ifs}\right|^2}\, df \tag{25}$$

[*cf.* Eqs. (9) and (5)]. Then the discrete convolution of Eq. (7) yields the Lagrange multipliers λ_{ht}, $t = 0, \ldots, M$. (The Levinson recursion also yields reflection coefficients $k_{h1}, \ldots, k_{hM}$. The condition that $|k_{ht}| < 1$ for all $t = 1, \ldots, M$ is a useful criterion for checking that $\boldsymbol{R}_h$ is a valid family of autocorrelations—i.e., that it admits an extension of positive type.)

There are inverted versions of the Levinson recursion that allow $\boldsymbol{R}_h$ to be computed from the family $\boldsymbol{a}_h = (a_{h0}, a_{h1}, \ldots, a_{hM})$ of inverse filter coefficients, either directly or via the reflection coefficients. These will also be of use.

Finally, to compute autocorrelation from the Lagrange multipliers, we could use the algorithms above together with an algorithm for computing inverse filter coefficients from the Lagrange multipliers. There are two procedures for this—one based on Cholesky factorization of a Toeplitz matrix [11] and one based on Newton's method [12,13]. We mention these although, as it turns out, we do not actually need to compute autocorrelations from Lagrange multipliers.

Partial Derivative Calculations

The computation of $\Delta \boldsymbol{R}_h$ and $\Delta \boldsymbol{\beta}$ requires the Jacobian matrices $\partial \boldsymbol{R}_h / \partial \boldsymbol{\lambda}_h$ with elements $\partial R_{ht} / \partial \lambda_{hu}$ $(t,u = 0, \ldots, M)$. Differentiation of Eq. (9) yields

$$\frac{\partial R_{ht}}{\partial \lambda_{hu}} = \int_0^{1/2} \frac{-\cos 2\pi ft \cos 2\pi fu}{\left| \sum_{v=0}^{M} \lambda_{hv} \cos 2\pi fv \right|^2} \, df.$$

Since the numerator of the integrand is equal to $-\frac{1}{2} \cos 2\pi f(t+u) - \frac{1}{2} \cos 2\pi f|t-u|$, we may write

$$\frac{\partial R_{ht}}{\partial \lambda_{hu}} = -S_{h,t+u} - S_{h,|t-u|}, \tag{26}$$

where

$$S_{hv} = \tfrac{1}{2} \int_0^{1/2} \frac{\cos 2\pi fv}{\left| \sum_{t=0}^{M} \lambda_{ht} \cos 2\pi ft \right|^2} \, df,$$

$v = 0, \ldots, 2M$. We express this in terms of the inverse filter coefficients a_{ht} and gain E_h corresponding to the Lagrange multipliers λ_{ht}. Equation (7) [*cf.* Eqs. (5) and (6)] implies that

$$\frac{1}{\left| \sum_{t=0}^{M} \lambda_{ht} \cos 2\pi ft \right|} = \frac{2E_h}{\left| \sum_{t=0}^{M} a_{ht} e^{2\pi ift} \right|^2}.$$

It follows that

$$S_{hv} = \int_0^{1/2} \frac{2E_h^2 \cos 2\pi fv}{\left| \sum_{t=0}^{M} a_{ht} e^{2\pi ift} \right|^4} \, df.$$

Regarding the a_{ht} $(t = 0, \ldots, M)$ as coefficients of a polynomial $A(z) = 1 + a_{h1}z + \cdots + a_{hM}z^M$, we compute the coefficients b_{hv} of the polynomial $A(z)^2 = 1 + b_{h1}z + \cdots + b_{h,2M}z^{2M}$,

$$b_{hv} = \sum_t a_{ht} a_{h,v-t}, \tag{27}$$

$v = 0, \ldots, 2M$. (The sum runs from $t = 0$ to v when $v \leqslant M$ and from $t = v - M$ to M when $v \geqslant M$.) Then

$$S_{hv} = \int_0^{1/2} \frac{2E_h^2 \cos 2\pi fv}{\left| \sum_{w=0}^{2M} b_{hw} e^{2\pi ifw} \right|^2} \, df.$$

Comparing this with Eq. (25), we see that the vector $\boldsymbol{S}_h$ depends on E_h^2 and the vector $\boldsymbol{b}_h$ in the same way that the vector $\boldsymbol{R}_h$ of autocorrelations depends on E_h and $\boldsymbol{a}_h$. Thus, any algorithm designed to compute autocorrelations from gain and inverse filter coefficients may also be used to compute the S_{hv} from the squared gain and the coefficients b_{hv}.

Here, then, is the procedure for computing $\partial \boldsymbol{R}_h / \partial \boldsymbol{\lambda}_h$. Start with E_h and the vector $\boldsymbol{a}_h$ of inverse filter coefficients; these will be available as a result of their use in the computation of $\boldsymbol{\lambda}_h$ from $\boldsymbol{R}_h$. Obtain the elements of $\boldsymbol{b}_h$ by Eq. (27). Compute $\boldsymbol{S}_h$ from E_h^2 and $\boldsymbol{b}_h$ as just described. Finally, use Eq. (26) to obtain $\partial \boldsymbol{R}_h / \partial \boldsymbol{\lambda}_h$.

Calculation of $\Delta \boldsymbol{R}_h$ and $\Delta \boldsymbol{\beta}$

We next show how to calculate $\Delta \boldsymbol{R}_h$ and $\Delta \boldsymbol{\beta}$ from the Jacobian matrices $\partial \boldsymbol{R}_h / \partial \boldsymbol{\lambda}_h$. We define the abbreviation $\mathbf{J}_h$ for the Jacobians by

$$J_{htu} = \partial R_{ht} / \partial \lambda_{hu},$$

$t,u = 0, \ldots, M$. We will also use a rectangular matrix $\mathbf{D}$ defined by

$$D_{rt} = \begin{cases} 1, & r = t, \\ 0, & r \neq t, \end{cases}$$

$r = 0, \ldots, m$; $t = 0, \ldots, M$. In terms of $\mathbf{D}$ and its transpose $\mathbf{D}'$, we can express quantities such as $\mathbf{D}\boldsymbol{R}_h$, the result of truncating the vector $\boldsymbol{R}$ to its first $m+1$ elements; $\mathbf{D}'\boldsymbol{\beta}$, the result of extending the vector $\boldsymbol{\beta}$ to length $M+1$ by appending zeros; and $\mathbf{D}\mathbf{J}_h\mathbf{D}'$, the $(m+1) \times (m+1)$ submatrix in the upper left corner of $\mathbf{J}_h$. We can thus rewrite Eqs. (8) and (10) as

$$\boldsymbol{\lambda}_h = \boldsymbol{\lambda}_h^0 + \mathbf{D}'\boldsymbol{\beta}$$

and

$$\mathbf{D}\sum_h \boldsymbol{R}_h = \boldsymbol{R}^{\text{tot}}.$$

After replacement of $\boldsymbol{\beta}$ and the $\boldsymbol{R}_h$ by $\boldsymbol{\beta} + \Delta\boldsymbol{\beta}$ and $\boldsymbol{R}_h + \Delta\boldsymbol{R}_h$, with corresponding replacement of $\boldsymbol{\lambda}_h$, we wish Eqs. (8) and (10) to hold, at least to a better approximation than before. Although $\boldsymbol{\lambda}_h$ and $\boldsymbol{R}_h$ are nonlinearly related by Eq. (9), the replacement value for $\boldsymbol{\lambda}_h$, for small changes, is given by $\boldsymbol{\lambda}_h + \Delta\boldsymbol{\lambda}_h$, where $\Delta\boldsymbol{\lambda}_h$ approximately satisfies

$$\mathbf{J}_h \Delta\boldsymbol{\lambda}_h = \Delta\boldsymbol{R}_h. \tag{28}$$

(That is, $\sum_{u=0}^{M} (\partial R_{ht}/\partial \lambda_{hu}) \Delta\lambda_{hu} = \Delta R_{ht}$, $t = 0, \ldots, M$.) To establish Eqs. (8) and (10) through the replacement, we wish

$$\boldsymbol{\lambda}_h + \Delta\boldsymbol{\lambda}_h = \boldsymbol{\lambda}_h^0 + \mathbf{D}'(\boldsymbol{\beta} + \Delta\boldsymbol{\beta}) \tag{29}$$

and

$$\mathbf{D}\sum_h (\boldsymbol{R}_h + \Delta\boldsymbol{R}_h) = \boldsymbol{R}^{\text{tot}} \tag{30}$$

to hold. By Eqs. (28) and (29), $\Delta\boldsymbol{R}_h$ should (approximately) satisfy

$$\Delta\boldsymbol{R}_h = \mathbf{J}_h(\boldsymbol{\lambda}_h^0 + \mathbf{D}'\boldsymbol{\beta} - \boldsymbol{\lambda}_h) + \mathbf{J}_h\mathbf{D}'\Delta\boldsymbol{\beta}. \tag{31}$$

Using this equation to eliminate $\Delta\boldsymbol{R}_h$ from Eq. (30), we obtain

$$\mathbf{D}\sum_h \boldsymbol{R}_h + \mathbf{D}\sum_h \mathbf{J}_h(\boldsymbol{\lambda}_h^0 + \mathbf{D}'\boldsymbol{\beta} - \boldsymbol{\lambda}_h) + \sum_h \mathbf{D}\mathbf{J}_h\mathbf{D}'\Delta\boldsymbol{\beta} = \boldsymbol{R}^{\text{tot}};$$

hence $\Delta\boldsymbol{\beta}$ may be computed by

$$\Delta\boldsymbol{\beta} = \left[\sum_h \mathbf{D}\mathbf{J}_h\mathbf{D}'\right]^{-1}\{\boldsymbol{R}^{\text{tot}} - \mathbf{D}\sum_h [\boldsymbol{R}_h + \mathbf{J}_h(\boldsymbol{\lambda}_h^0 + \mathbf{D}'\boldsymbol{\beta} - \boldsymbol{\lambda}_h)]\}. \tag{32}$$

The procedure for computing $\Delta\boldsymbol{\beta}$ and $\Delta\boldsymbol{R}_h$, in summary, is first to compute the quantities $\mathbf{J}_h(\boldsymbol{\lambda}_h^0 + \mathbf{D}'\boldsymbol{\beta} - \boldsymbol{\lambda}_h)$, then obtain $\Delta\boldsymbol{\beta}$ from Eq. (32), and finally obtain the $\Delta\boldsymbol{R}_h$ from Eq. (31).

Summary of Second Algorithm

We have discussed various parts of the algorithm; here is how they fit together. Inputs are the prior autocorrelation estimates for the individual signal components h and the measured autocorrelations for their sum:

$$\boldsymbol{R}_h^0 = (R_{\cdot}^{\cdot} \; R_{h1}^0 \; , \; \ldots \; , \; R_{hM_h}^0)$$

and

$$\boldsymbol{R}^{\text{tot}} = (R_0^{\text{tot}}, R_1^{\text{tot}} \; , \; \ldots \; , \; R_m^{\text{tot}}).$$

No initial assumptions are made about the orders M_h of the prior estimates.

The steps of the second algorithm are as follows:

Step 1. Define

$$M = \max\{m, M_1 \; , \; \ldots \; , \; M_p\}.$$

Step 2. By the Levinson recursion, compute families of inverse filter coefficients $(a_{h0}=1, a_{h1}, \ldots, a_{hM_h})$ and reflection coefficients $(k_{h1}, \ldots, k_{hM_h})$ and a gain E_h corresponding to $\boldsymbol{R}_h^0$ for each h. (The conditions $|k_{ht}| < 1$ may be tested as a validity check for the inputs.) Define $a_{ht} = k_{ht} = 0$ when $M_h < t \leqslant M$. The results are two vectors $\boldsymbol{a}_h = (a_{h0}=1, a_{h1}, \ldots, a_{hM})$ and $\boldsymbol{k}_h = (k_{h1}, \ldots, k_{hM})$ and a gain E_h for each h.

Step 3. Compute the families of Lagrange multipliers

$$\boldsymbol{\lambda}_h^0 = (\lambda_{h0}^0, \lambda_{h1}^0, \ldots, \lambda_{hM}^0)$$

corresponding to $\boldsymbol{R}_h^0$ from $\boldsymbol{a}_h$ and E_h for each h by Eq. (7) (with $\boldsymbol{\lambda}_h^0$ in place of $\boldsymbol{\lambda}_h$).

Step 4. (Initialize variables.) By a standard LPC algorithm, determine a family $(R_{h0}, R_{h1} \; , \; \ldots \; , \; R_{hM})$ of autocorrelations corresponding to the gain and reflection coefficients E_h and $\boldsymbol{k}_h$ for each h; use these as initial values for the variables $\boldsymbol{R}_h$. (This results in

$$R_{ht} = R_{ht}^0$$

for $0 \leqslant t \leqslant M_h$; hence computing $\boldsymbol{R}_h$ from E_h and $\boldsymbol{k}_h$ is actually unnecessary unless $M_h < M$.) Also, set

$$\boldsymbol{\lambda}_h \leftarrow \boldsymbol{\lambda}_h^0$$

and

$$\beta_t \leftarrow 0, \qquad t = 0, 1 \; , \; \ldots \; , \; m.$$

Step 5. (Beginning of main loop.) Compute coefficients

$$\boldsymbol{b}_h = (b_{h0}=1, b_{h1} \; , \; \ldots \; , \; b_{h,2M})$$

from $\boldsymbol{a}_h$ for each h by Eq. (27).

Step 6. Compute the quantities

$$\boldsymbol{S}_h = (S_{h0}, S_{h1} \; , \; \ldots \; , \; S_{h,2M})$$

for each h by an algorithm for computing autocorrelations from inverse filter coefficients and gains. As inputs, let the family $\boldsymbol{b}_h$ of coefficients from Step 5 play the role of inverse filter coefficients and the squared power E_h^2 play the role of gain. The family $\boldsymbol{S}_h$ comprises the resulting "autocorrelations." Define the Jacobian matrices $\mathbf{J}_h$ in terms of $\boldsymbol{S}_h$ by Eq. (26).

Step 7. Compute the quantities $\mathbf{J}_h(\boldsymbol{\lambda}_h^{(0)} + \mathbf{D}'\boldsymbol{\beta} - \boldsymbol{\lambda}_h)$. Define $\Delta\boldsymbol{\beta}$ by Eq. (32), and replace $\boldsymbol{\beta}$ by a new value,

$$\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} + \Delta\boldsymbol{\beta}.$$

Obtain a tentative value for $\Delta\boldsymbol{R}_h$ for each h from Eq. (31). (At this point the constraint equation [Eq. (30)] should be satisfied.)

Step 8. Check that the sums $\boldsymbol{R}_h + \Delta\boldsymbol{R}_h$ are valid families of autocorrelations. We can do this by using the Levinson recursion to compute reflection coefficients and checking that these are less than 1 in magnitude; it is also necessary to check that each R_{h0} is positive. If $\boldsymbol{R}_h + \Delta\boldsymbol{R}_h$ fails the test for any h, then reduce the $\Delta\boldsymbol{R}_h$ for all h by a constant factor. Repeat the testing and reduction until the $\boldsymbol{R}_h + \Delta\boldsymbol{R}_h$ pass the test. Then replace the $\boldsymbol{R}_h$ by new values

$$\boldsymbol{R}_h \leftarrow \boldsymbol{R}_h + \Delta\boldsymbol{R}_h$$

for each h.

Step 9. Decide whether to continue. If the new value of $\boldsymbol{R}_h$ (computed in Step 8) is not equal to the previous value for each h to within a specified relative tolerance, the main loop will have to be repeated from Step 5. Even if the new $\boldsymbol{R}_h$ *are* close enough to the previous values, the main loop should be repeated if $\boldsymbol{R}_h$ had to be reduced in Step 8. If the loop *is* to be repeated, continue with Step 10 before going back to Step 5. If results from Step 10 are required for output, do Step 10 before exiting. Otherwise terminate the algorithm now.

Step 10. At this point new values for the $\boldsymbol{R}_h$ from Step 8 are available. Corresponding new values for the quantities $\boldsymbol{a}_h$, E_h, and $\boldsymbol{\lambda}_h$ will be needed if there is to be another iteration of the main loop. They may also be desired as outputs from the algorithm. Nothing further need be done, however, if there is not to be another iteration, and if the only outputs desired are the posterior autocorrelation estimates. Values for $\boldsymbol{a}_h$ and E_h may already be available as results of the use of the Levinson recursion in Step 8; if not, they should be computed now. To obtain $\boldsymbol{\lambda}_h$ from $\boldsymbol{a}_h$ and E_h, use Eq. (7). Then either go back to Step 5 or terminate the algorithm, whichever is called for.

DISCUSSION

Both algorithms have been implemented in APL and in FORTRAN. The APL implementations were written first for ease of programming and debugging. The APL functions were then converted to FORTRAN for incorporation in a commercial signal-processing package that we use for experimental work with speech. Both versions of the algorithms are available from the author.

The second algorithm is so much faster than the first, that the first can be recommended only for applications that really demand the generality of unequally spaced lags, unequally spaced frequencies, or completely arbitrary prior spectral shapes. The FORTRAN version of the first algorithm, running on a VAX 11/750* with floating-point accelerator, uses about 18.5 s of cpu (central-processing-unit) time per frame of speech in a typical speech-processing application—a two-signal analysis with prior spectral estimates given at 128 equispaced frequencies and autocorrelation constraints given at 13 lags: 0, 1, ..., 12. The FORTRAN version of the second algorithm, running on the same machine, uses

*Trademark of Digital Equipment Corporation.

about 2.0 s per frame on the same problem, where the prior spectral estimates are 12^{th}-order all-pole spectra specified by autocorrelation values at lags 0, 1 , . . . , 12.

The second algorithm also uses less space than the first. In terms of the number p of signals, the number n of discrete frequencies, and the order m of the autocorrelation constraints, the FORTRAN implementation of the first algorithm uses array storage of $3(m+1)n + (m+1)^2 + 3pn + 3n + 4(m+1)$ floating-point (or double-precision) locations, $2(m+1)$ integer locations, and an additional amount independent of the dimensions of the inputs. For most applications, the dominant term is $3(m+1)n$, which corresponds to storage for the matrices **C**, **N**, and $\mathbf{N}'^{t}$ of Eqs. (2), (18), and (23). In terms of p, m, and the order M of the all-pole posterior spectra, the second algorithm uses $10(M+1)p + (M+1)^2 + 7(M+1) + 5(m+1)$ floating-point locations, $2(m+1)$ integer locations, and a fixed additional amount. Typical values of the size parameters are $p=2$, $n=128$, and $m=M=12$. With these values, the first algorithm uses 6365 floating-point locations, as compared with 585 for the second. Of the 6365 locations, nearly 80% are occupied by the three matrices that account for the dominant term $3(m+1)n$.

We do not claim to have produced optimally efficient multisignal MCESA algorithms; there may exist algorithms much superior to the two presented here in their respective domains. The efficiency of these algorithms should not, therefore, be taken as a measure of the intrinsic efficiency of multisignal MCESA. The second algorithm, however, is adequate to permit considerable experimentation with speech noise reduction, and it interfaces well with conventional LPC analysis/synthesis software. It has been possible to process entire sentences of speech corrupted with real or synthetic additive noise, to synthesize speech from the posterior speech autocorrelation estimates, and to listen to the results. Effective noise suppression has been demonstrated in experiments with a variety of methods for choosing prior speech and noise estimates and estimating autocorrelations from the sum of the speech and the noise signals [3].

ACKNOWLEDGMENTS

I thank J. Shore for suggesting in conversation the possibility of an algorithm like the second and for helpful comments on a draft of this report. I (with others) am grateful to J. Buck for writing the FORTRAN implementations of the algorithms.

REFERENCES

1. R.W. Johnson and J.E. Shore, "Minimum-Cross-Entropy Spectral Analysis of Multiple Signals," *IEEE Trans. Acoust., Speech, Signal Process.* **ASSP-31** (June 1983), in press.

2. J.E. Shore, "Minimum-Cross-Entropy Spectral Analysis," *IEEE Trans. Acoust., Speech, Signal Process.* **ASSP-29**, 230-237 (April 1981).

3. R. Johnson, J.E. Shore, J. Buck, and D. Burton, "Speech Noise Reduction by Means of Multisignal Minimum-Cross-Entropy Spectral Analysis," in *Proceedings ICASSP83,* Boston, Mass., April 1983, pp. 1129-1132.

4. J.P. Burg, "Maximum Entropy Spectral Analysis," presented at the 37th Annual Meeting of the Society of Exploration Geophysicists, Oklahoma City, Okla., 1967.

5. J.P. Burg, "Maximum Entropy Spectral Analysis," Ph.D. dissertation, Stanford University, 1975 (University Microfilms No. AAD75-25, 499).

6. J.D. Markel and A.H. Gray, Jr., *Linear Prediction of Speech,* Springer-Verlag, New York, 1976.

7. R. Penrose, "A Generalized Inverse for Matrices," *Proc. Cambridge Philos. Soc.* **51**, 406-413 (1955).

8. M.A. Jenkins, "The Solution of Linear Systems of Equations and Linear Least Squares Problems in APL," Technical Report 320-2989, IBM New York Scientific Center, New York, N.Y., 1970.

9. Digital Signal Processing Committee, IEEE Acoustics, Speech, and Signal Processing Society, eds., *Programs for Digital Signal Processing*, IEEE Press, New York, 1979.

10. N. Levinson, "The Wiener RMS (Root Mean Square) Error Criterion in Filter Design and Prediction," *J. Math. Phys.* **25**, 261-278 (1946).

11. F.L. Bauer, "Ein direktes Iterationsverfahren zur Hurwitz-Zerlegung eines Polynoms," *Arch. Electr. Übertrag.* **9**, 285-290 (1955).

12. G. Wilson, "Factorization of the Covariance Generating Function of a Pure Moving Average Process," *SIAM J. Numer. Anal.* **6**, 1-7 (1969).

13. D.P. Laurie, "Efficient Implementation of Wilson's Algorithm for Factorizing a Self-Reciprocal Polynomial," *BIT* **20**, 257-259 (1980).

Appendix A

APL PROGRAM FOR FIRST ALGORITHM

The argument P of the APL function *MCESP2P* below corresponds to $\boldsymbol{f}$ and $\mathbf{P}$ in the conventional notation. The argument R corresponds to $\boldsymbol{t}$ and $\boldsymbol{R}^{\text{tot}}$, and the result Q corresponds to $\boldsymbol{f}$ and $\mathbf{Q}$. P is a matrix with frequencies f_k in row 1; each additional row contains prior spectral estimates P_{hk} for one signal component h. R is a two-rowed matrix with lags t_r in row 1 and autocorrelations R_r^{tot} in row 2. Q is a matrix with the same shape and format as P: the frequencies f_k in row 1 and posterior spectral estimates Q_{hk} in the remaining rows.

MCESP2PW is a version of the program that implements a generalization of multisignal MCESA that is described elsewhere [A1]. The argument P has an additional first column containing "relative weights." The first element of the column is ignored. Each other element of the first column is a weighting factor associated with the prior spectral estimate in the rest of the same row. The format is

$$\begin{bmatrix} * & f_1 & \cdots & f_n \\ w_1 & P_{11} & \cdots & P_{1n} \\ & & \cdots & \\ w_p & P_{p1} & \cdots & P_{pn} \end{bmatrix}$$

The argument R and the result Q are as for *MCESP2P*; Q has no additional first column.

REFERENCE

A1. R.W. Johnson and J.E. Shore, "Multisignal Minimum-Cross-Entropy Spectrum Analysis with Weighted Priors," NRL Report 8731, in publication; submitted to *IEEE Trans. Acoust., Speech, Signal Process.*

```
     ∇ Q←P MCESP2P R;C;F;X;L;Q0;ΔL
[1]    C←2×2○(R[1;]×○2)∘.×F←P[1;]
[2]    R←R[2;]
[3]    P← 1 0 ↓P
[4]    X←R+.×⌹⍉C
[5]    L←(⍴R)⍴0
[6]    P←↕Q←P
[7]   A:Q0←Q
[8]    R←(+⌿Q*2)*0.5
[9]    ΔL←(((+⌿Q)-X)÷R)⌹⍉C×(⍴C)⍴R
[10]  B:Q←↕P+(⍴P)⍴(L+ΔL)+.×C
[11]   →(∧/,Q>0)/C
[12]   ΔL←ΔL×0.9÷1-⌊/,Q0÷Q
[13]   →B
[14]  C:L←L+ΔL
[15]   →(∨/,Q≠Q0)/A
[16]   Q←F,[1] Q
     ∇
```

```
     ∇ Q←P MCESP2PW R;W;F;C;X;L;Q0;ΔL
[1]    W←↕1↓P[;1]
[2]    F←1↓P[1;]
[3]    C←2×2○(R[1;]×○2)∘.×F
[4]    R←R[2;]
[5]    P← 1 1 ↓P
[6]    X←R+.×⌹⍉C
[7]    L←(⍴R)⍴0
[8]    P←↕Q←P
[9]   A:Q0←Q
[10]   R←(W+.×Q*2)*0.5
[11]   ΔL←(((+⌿Q)-X)÷R)⌹⍉C×(⍴C)⍴R
[12]  B:Q←↕P+W∘.×(L+ΔL)+.×C
[13]   →(∧/,Q>0)/C
[14]   ΔL←ΔL×0.9÷1-⌊/,Q0÷Q
[15]   →B
[16]  C:L←L+ΔL
[17]   →(∨/,Q≠Q0)/A
[18]   Q←F,[1] Q
     ∇
```

Appendix B

FORTRAN PROGRAM FOR FIRST ALGORITHM

The following FORTRAN subroutine DOMC2 implements a two-signal version of the first algorithm. It will handle autocorrelations of order up to 29 and up to 513 discrete frequencies. The array C should be initialized in the calling program by

```
      DO 1 I=1,M
      DO 1 K=1,N
1     C(I,K) = 2 * COS(2 * PI * T(I) * F(K))
```

before the first call on DOMC2. It need not be reinitialized for subsequent calls as long as the same sets of frequencies F(K) and lags T(I) are to be used. The input argument WT ("relative weights") is concerned with a generalization of multisignal MCESA that is described elsewhere [B1]; if WT(1) and WT(2) are made equal and positive (say both equal to 1.0), the algorithm will behave as here described.

DOMC2 requires a function DISTD and a subroutine MPD, both shown below. DISTD defines the stopping criterion.

MPD defines its argument INV as the Moore-Penrose generalized inverse of the transpose of its argument C, provided that the rows of C are linearly independent. Lawson and Hanson [B2] give general-inverse routines that are numerically superior, though more complex. MPD uses a matrix inversion subroutine MINVD, not shown. The form of the call is CALL MINVD(A,N,D,L,M), where A is an N-by-N double-precision matrix, and L and M are integer scratch vectors of length N. (A determinant may be returned through D but is ignored in MPD.) A contains the input matrix, which is destroyed and replaced by the inverse.

REFERENCES

B1. V. Johnson and J. E. Shore, "Multisignal Minimum-Cross-Entropy Spectrum Analysis with ghted Priors," NRL Report 8731, in publication; submitted to *IEEE Trans. Acoust., Speech, Signal Process.*

B2. C.L. Lawson and R.J. Hanson, *Solving Least-Squares Problems,* Prentice-Hall, Englewood Cliffs, N.J., 1974.

```
      SUBROUTINE DOMC2(C,R,P,M,N,WT,B,Q,INV,N1,IERR,LUN)
      REAL*8 R(30),B(30),DB(30),P(513,2),Q(513,2),W(513),X(513)
      REAL*8 C(M,N),INV(M,N),N1(M,N),TMP(30,30),Q0(513,2),WT(2)
      REAL*8 AVG,D,QSUM,QSUMSQ,RMI513,SCALE
      INTEGER K1(30),K2(30)
      LOGICAL FLAG
C SUBROUTINE TO PERFORM MINIMUM CROSS-ENTROPY SPECTRAL ANALYSIS
C ON MULTIPLE SPECTRA
C DOUBLE PRECISION VERSION
C
C       NAVAL RESEARCH LABORATORY
C       J. T. BUCK
C       ADAPTED FROM APL VERSION BY R. JOHNSON
C
C INPUT ARGUMENTS:
C       C - ARRAY OF COSINE TERMS - 2*COS(2*PI*T(I)*F(K))
C         - FOR TIME AND FREQUENCY POINTS
C       R - AUTOCORRELATIONS
C       P - PRIOR SPECTRAL ESTIMATE
C               FOR EACH OF 2 SPECTRA
C       M - NUMBER OF TIME POINTS
C       N - NUMBER OF FREQUENCY POINTS
C       WT- RELATIVE WEIGHTS OF PRIOR SPECTRA
C       LUN IF > 0, LUN FOR STATUS MESSAGES
C OUTPUT ARGUMENTS:
C       B - LAGRANGE MULTIPLIERS USED IN SOLUTION
C       Q - MINIMUM CROSS-ENTROPY SPECTRAL ESTIMATES FOR EACH SPECTRUM
C       IERR - ERROR FLAG - 0 IF OK, NONZERO OTHERWISE
C
C SCRATCH ARGUMENTS:
C       N1 - WORK MATRIX
C       INV - GETS GENERALIZED INVERSE OF WORK MATRIX
C       W,X - WORK VECTORS
C       Q0 - SAVES PREVIOUS VALUE OF Q
C       DB - INCREMENT FOR CALCULATING NEXT B
C       TMP,K1,K2 - TEMPORARIES REQUIRED FOR MPD ROUTINE
C
C SUBROUTINES REQUIRED
C       MPD - CALCULATES GENERALIZED INVERSE
C       DISTD - DISTANCE MEASURE
C       MINVD - INVERTS A MATRIX (CALLED BY MPD)
C
      IERR=0
C GET AVG VALUE OF PRIORS (FOR CONVERGENCE CHECK)
      AVG=0.0
      DO I=1,N
        DO J=1,2
          AVG=AVG+P(I,J)
        ENDDO
      ENDDO
      AVG=AVG/(N*2)
C INITIALIZE B,C,AND X
      DO J=1,M
        B(J)=0.0
      ENDDO
```

```
        CALL MPD(C,INV,M,N,TMP,K1,K2)
        DO K=1,N
          DO L=1,2
            Q(K,L)=P(K,L)
            P(K,L)=1.0/P(K,L)
          ENDDO
          X(K)=0.0
          DO J=1,M
            X(K)=X(K)+INV(J,K)*R(J)
          ENDDO
        ENDDO
        NPASS=0
        NRSOJ=0
C MAIN LOOP STARTS HERE
30      NPASS=NPASS+1
        IF(NPASS.GT.100)THEN
C Convergence failure - type message, return error
          D=DISTD(Q,Q0,N*2)/AVG
          TYPE 31,D
31        FORMAT(' Convergence failed after 100 passes - D= ',1PE13.6)
          IERR=1
          RETURN
        ENDIF
C
        DO K=1,N
C COPY OLD VALUE OF Q, CALC AVG, MEAN SQUARE VALUES
          QSUM=0.0
          QSUMSQ=0.0
          DO L=1,2
            Q0(K,L)=Q(K,L)
            QSUM=QSUM+Q(K,L)
            QSUMSQ=QSUMSQ+Q(K,L)**2/WT(L)
          ENDDO
          QSUMSQ=SQRT(QSUMSQ)
          W(K)=(QSUM-X(K))/QSUMSQ
          DO J=1,M
            N1(J,K)=C(J,K)*QSUMSQ
          ENDDO
        ENDDO
C GET INVERSE OF WORK MATRIX
        CALL MPD(N1,INV,M,N,TMP,K1,K2)
C CALCULATE INCREMENT FOR LAGRANGE MULTIPLIERS
        DO J=1,M
          DB(J)=0.0
          DO K=1,N
            DB(J)=DB(J)+INV(J,K)*W(K)
          ENDDO
        ENDDO
C Calculate new Q. Check for overshoot. Loop until no overshoot occurs.
        FLAG=.TRUE.
        DO WHILE(FLAG)
          FLAG=.FALSE.
C Calculate new Q's from P's and B's. If any are negative, set FLAG
          DO K=1,N
            DO L=1,2
```

```
              Q(K,L)=P(K,L)
              DO J=1,M
                Q(K,L)=Q(K,L)+(B(J)+DB(J))*C(J,K)/WT(L)
              ENDDO
              Q(K,L)=1.0/Q(K,L)
              IF(Q(K,L).LT.0.0)FLAG=.TRUE.
            ENDDO
          ENDDO
C If overshoot occured (resulting in negative sprectral power)
          IF (FLAG) THEN
C Then scale the increment and repeat
            RMIN=1.E37
            DO K=1,N
              DO L=1,2
                IF(Q0(K,L)/Q(K,L).LT.RMIN)RMIN=Q0(K,L)/Q(K,L)
              ENDDO
            ENDDO
            SCALE=0.9/(1.0-RMIN)
            NRSOJ=NRSOJ+1
            DO J=1,M
              DB(J)=DB(J)*SCALE
            ENDDO
          ENDIF
        ENDDO
C Calculate new b value,test for termination
        DO J=1,M
          B(J)=B(J)+DB(J)
        ENDDO
        IF(DISTD(Q,Q0,N*2).LT.1.E-10*AVG)THEN
          IF(LUN.NE.0)WRITE(LUN,100)NPASS,NRSOJ
          RETURN
        ENDIF
100     FORMAT(12X,'Passes:',I3,'     Overshoots:',I3)
        GOTO 30
        END



        FUNCTION DISTD(A,B,N)
        REAL*8 A(N),B(N),DISTD
C
C       J. T. BUCK
C       NAVAL RESEARCH LABORATORY
C
C THIS ROUTINE CALCULATES A DISTANCE MEASURE BETWEEN THE VECTORS
C A AND B, WHICH IS SIMPLY THE MAXIMUM ABSOLUTE VALUE DIFFERENCE
C OF THE COMPONENTS.
        DISTD=-1.E37
        DO 10 I=1,N
        IF(ABS(A(I)-B(I)).GT.DISTD)DISTD=ABS(A(I)-B(I))
10      CONTINUE
        RETURN
        END
```

```
      SUBROUTINE MPD(C,INV,M,N,TMP,K1,K2)
C
C MPD: EMULATES APL DOMINO OPERATOR (GENERALIZED INVERSE)
C      DOUBLE PRECISION VERSION
C
C      J. T. BUCK
C      NAVAL RESEARCH LABORATORY
C
C INPUT ARGUMENTS:
C
C      C -     M BY N MATRIX TO INVERT
C      M, N -  MATRIX DIMENSIONS
C
C OUTPUT ARGUMENTS:
C
C      INV -   M BY N RESULT
C
C SCRATCH ARGUMENTS:
C
C      TMP -   M BY M REAL MATRIX
C      K1,K2 - INTEGER INDEX VECTORS, LENGTH M
C
C SUBROUTINES REQUIRED:
C
C      MINVD - INVERTS A MATRIX
C
      REAL*8 C(M,N),INV(M,N),TMP(M,M),D
      INTEGER K1(M),K2(M)
C
C THIS ROUTINE CALCULATES THE GENERALIZED INVERSE OF THE TRANSPOSE
C OF C AND RETURNS THE RESULT IN INV. THE ARGUMENTS TMP,K1,AND K2
C ARE USED BY MINVD, WHICH INVERTS A MATRIX.
C
C INV=INVERSE(C TIMES C-TRANSPOSE) TIMES C
C
C CALCULATE TMP=C TIMES C-TRANSPOSE
      DO 10 I=1,M
        DO 10 J=1,M
          TMP(I,J)=0.0
          DO 10 K=1,N
            TMP(I,J)=TMP(I,J)+C(I,K)*C(J,K)
10    CONTINUE
C GET THE INVERSE OF TMP
      CALL MINVD(TMP,M,D,K1,K2)
C CALCULATE INV= TMP TIMES C
      DO 20 I=1,M
        DO 20 J=1,N
          INV(I,J)=0.0
          DO 20 K=1,M
            INV(I,J)=INV(I,J)+TMP(I,K)*C(K,J)
20    CONTINUE
      RETURN
      END
```

Appendix C

APL PROGRAM FOR SECOND ALGORITHM

The arguments $R0$ and *RTOT* and the result R of the APL function *MCEAUTW* below correspond to the $\boldsymbol{R}_h^0$, $\boldsymbol{R}^{tot}$, and the $\boldsymbol{R}_h$ in conventional notation. $R0$ is a matrix with one row for each signal component h and one column for each autocorrelation lag from 0 through M_h (which is independent of h; the APL version requires all the priors to be of the same order). In addition, column 1 contains "relative weights," which are concerned with a generalization of multisignal MCESA that is described elsewhere [C1]. Thus the prior autocorrelation information occupies columns 2 through M_h+2 of $R0$. The user who is not concerned with relative weights may simply set all elements of the first column of $R0$ equal to 1. *RTOT* is a vector of length $m+1$. R is returned as a matrix with the same number of rows as $R0$ and one column for each lag from 0 through M (but no column of weights). *MCEAUTW* uses several other functions for discrete convolution and various LPC parameter conversions.

If the arguments X and Y of *CVR* are vectors of length M and length N, respectively, the result is their discrete convolution, a vector of length $M+N-1$. If X and Y are matrices with the same number of rows, the result is a matrix, also with the same number of rows; each row of the result contains the discrete convolution of the corresponding row of X with the corresponding row of Y. *CVR* similarly extends row by row to arrays with any number of dimensions. Thus *CVR* applied to a 2×3×4 array and a 2×3×5 array yields a 2×3×8 result.

The functions for LPC parameter conversions likewise extend systematically to higher-dimensional arguments, although we describe only their application to arguments of the lowest dimension. Thus *AUTΔPAR* is said to apply to a vector K and a scalar $R0$ to yield a vector result R. However, it is then to be understood that *AUTΔPAR* may also apply to a matrix K and a vector $R0$ with one element for each row of K to yield a matrix result R with one row for each row of K.

The argument to *PREΔPAR* is a vector K containing parcor coefficients, or negative reflection coefficients, $-k_1, \ldots, -k_M$. The result *PREΔPAR K* is a vector A containing inverse filter coefficients. The coefficient a_0 is absent, and the sign convention is the opposite of that used in the FORTRAN programs and the main body of this report. Thus the contents of A are $-a_1, \ldots, -a_M$.

The function *PARΔPRE* is an inverse to *PREΔPAR*. The argument is a vector A of inverse filter coefficients (with reversed sign), and the result *PREΔPAR A* is a vector K of negative reflection coefficients.

The function *LAGΔPRE* takes as arguments a vector A of inverse filter coefficients (with reversed sign) and a scalar gain E. The result *E LAGΔPRE A* is computed by Eq. (7), but without the factor of ½ in λ_0 (and we have suppressed the subscript h). The result is thus a vector B containing $2\lambda_0, \lambda_1, \ldots, \lambda_M$.

The arguments to *AUTΔPAR* are a vector K containing negative reflection coefficients and a scalar $R0$ containing a total power R_0. The result *R0 AUTΔPAR K* is a vector R containing autocorrelation values $R_0, R_1, \ldots, R_M$.

PARΔAUT, an inverse function to *AUTΔPAR*, takes R as its argument and returns K as its result *PARΔAUT R*.

GAIΔPAR yields a scalar result, the ratio of LPC gain to total power, or E/R_0 in the notation of Eq. (25) (with subscripts h suppressed). This result is computed as a function of negative reflection coefficients K.

REFERENCE

C1. R.W. Johnson and J.E. Shore, "Multisignal Minimum-Cross-Entropy Spectrum Analysis with Weighted Priors," NRL Report 8731, in publication; submitted to *IEEE Trans. Acoust., Speech, Signal Process.*

```
     ∇ R←R0 MCEAUTW RTOT;W;K;D;MU;M;F;I1;I2;A;E;L0;L;B;S;T;ΔR;J;ΔB;SCALED
[1]    W←↕R0[;1]
[2]    R0← 0 1 ↓R0
[3]    K←PARΔAUT R0
[4]    →(1∨.≤|(,K),PARΔAUT RTOT)/0
[5]    D←(⍴R0)[1]
[6]    MU←¯1+⍴RTOT
[7]    M←MU⌈¯1+(⍴R0)[2]
[8]    F←(D,M+1)⍴0.5,M⍴1
[9]    I1←0,⍳MU
[10]   I2←(I1+M+1)∘.-I1
[11]   I1←(I1+M+1)∘.+I1
[12]   A←PREΔPAR K
[13]   E←R0[;1]×GAIΔPAR K
[14]   L0←F×(D,M+1)↑E LAGΔPRE A
[15]   A←(D,M)↑A
[16]   L←L0
[17]   B←(M+1)⍴0
[18]   R←R0
[19]   →((⍴R0)[2]≥M+1)/A
[20]   R←R0[;1] AUTΔPAR(D,M)↑K
[21] A:K←PARΔPRE 0 1 ↓-(¯1,A) CVR ¯1,A
[22]   S←((E*2)↕GAIΔPAR K) AUTΔPAR K
[23]   S←(⌽ 0 1 ↓(0,-M)↓S),S
[24]   T←(0,M)↓(0,-M)↓S CVR⌽(L0+W∘.×B)-L
[25]   ΔR←-(⌽(0,-M)↓T)+(0,M)↓T
[26]   J←-W+.×S
[27]   ΔB←(RTOT-(MU+1)↑+⌿R+ΔR)⌸J[I1]+J[I2]
[28]   B←B+(M+1)↑ΔB
[29]   T←(0,MU)↓(0,-M)↓S CVR W∘.×⌽ΔB
[30]   ΔR←ΔR-(⌽(0,-M)↓T)+(0,M)↓T
[31]   SCALED←0
[32]   R0←R
[33]   →((⌈/|RTOT-+⌿R0+ΔR)≤RTOT[1]×1E¯5)/B
[34]   'CONSTRAINTS VIOLATED'
[35]   →0
[36] B:R←R0+ΔR
[37]   K←PARΔAUT R
[38]   →((0∧.<R[;1])∧1∧.>|,K)/C
[39]   ΔR←ΔR×0.75
[40]   SCALED←1
[41]   →B
[42] C:→(SCALED)/D
[43]   →(∧/,R=R0)/0
[44] D:A←PREΔPAR K
[45]   E←R[;1]×GAIΔPAR K
[46]   L←F×E LAGΔPRE A
[47]   →A
     ∇
```

```
     ∇ Z←Y CVR X;M;N;R
[1]    M←¯1↑⍴X
[2]    N←¯1↑⍴Y
[3]    R←⍴⍴X
[4]    Z←+/[R]((⍴X)⍴1-⍳M)⌽((⍴X),M+N-1)↑(((R+1),⍳R)⍉(N,⍴X)⍴X)×
       (R,(⍳R-1),R+1)⍉(M,⍴Y)⍴Y
     ∇
```

```
     ∇ A←PRE∆PAR K;D;M;N;I
[1]    D←⍴1/K
[2]    M←×/¯1↓D
[3]    N←¯1↑D
[4]    K←(M,N)⍴K
[5]    A←0/K
[6]    →(N<I←1)/B
[7]   A:A←(A-K[;(I-1)⍴I]×⌽A),K[;I]
[8]    →(N≥I←I+1)/A
[9]   B:A←D⍴A
     ∇
```

```
     ∇ K←PAR∆PRE A;D;M;N;L
[1]    D←⍴1/A
[2]    M←×/¯1↓D
[3]    N←¯1↑D
[4]    K←(M,N)⍴A
[5]   A:→(0≥N←N-1)/B
[6]    L←K[;N+1]
[7]    K[;⍳N]←(K[;⍳N]+K[;⌽⍳N]×⍉(N,M)⍴L)↑⍉(N,M)⍴1-L*2
[8]    →A
[9]   B:K←D⍴K
     ∇
```

```
     ∇ B←E LAG∆PRE A;N;M
[1]    A←¯1,A
[2]    N←¯1↑⍴A
[3]    M←¯1↑⍴A
[4]    B←(M,0)⍴0
[5]   A:B←B,+/((M,N)↑A)×(M,-N)↑A
[6]    →(0<N←N-1)/A
[7]    B←B↑⍉(⌽⍴A)⍴⍉E
     ∇
```

```
     ∇ R←R0 AUT∆PAR K;D;M;N;A;EK;I
[1]    D←ρ1/K
[2]    M←×/¯1↓D
[3]    N←¯1↑D
[4]    EK←(M,N)ρK××\((-ρD)↑¯1)↓R0,1-K*2
[5]    K←(M,N)ρK
[6]    R←A←(M,0)ρ0
[7]    →(N<I←1)/B
[8]   A:R←R,EK[;I]++/A×ΦR
[9]    A←(A-K[;(I-1)ρI]×ΦA),K[;I]
[10]   →(N≥I←I+1)/A
[11]  B:R←R0,DρR
     ∇


     ∇ K←PAR∆AUT R;E;B;KK
[1]    R←⍉R
[2]    K←0≠R
[3]    →(1≥1↑ρR)/C
[4]    E←(1↓ρR)ρR
[5]    B←0,[0.5](ρE)ρ1
[6]    →B
[7]   A:E←E×1-KK*2
[8]    B←0,[1] B-(⊖B)×(ρB)ρKK
[9]   B:KK←(+≠B×(ρB)↑R)⌿E
[10]   K←K,[1] KK
[11]   →((1↑ρB)<1↑ρR)/A
[12]  C:K←⍉K
     ∇


     ∇ G←GAI∆PAR K
[1]    G←×/1-K*2
     ∇
```

Appendix D

FORTRAN PROGRAM FOR SECOND ALGORITHM

The following FORTRAN subroutine MCLPC implements a one- or two-signal version of the second algorithm and will handle autocorrelations of order 29 or less. The input argument W (relative weights) is concerned with a generalization of multisignal MCESA that is described elsewhere [D1]; if W(1) and W(2) are made equal and positive (say both equal to 1.0), the algorithm will behave as here described.

MCLPC requires a matrix-inversion subroutine MINVD and subroutines DLEVIN, DA2L, DRC2AU, and DA2RC for conversions among various sets of LPC parameters.

The form of a call for MINVD is CALL MINVD (A,N,D,L,M), where A is an N-by-N double-precision matrix and L and M are integer scratch vectors of length N. (A determinant may be returned through D but is ignored in MCLPC.) A contains the input matrix, which is destroyed and replaced by the inverse.

DLEVIN implements the standard Levinson recursion. A call for DLEVIN has the form CALL DLEVIN (MP,R,A,ALPHA,RC), where R, A, and RC are double-precision vectors, ALPHA is a double-precision scalar, and MP is an integer. The inputs are MP, which is the order plus one, and R, which contains MP autocorrelation values from lag 0 to lag M=MP−1. The outputs are MP inverse filter coefficients in A, M reflection coefficients in RC, and the gain (excitation power) in ALPHA. A(1) contains $a_0 = 1$. The subroutine body is adapted from Markel and Gray's subroutine AUTO [D2, p. 51] with the following changes: convert to double precision; redimension local arrays to 60 (the subroutines must handle twice the order of the input autocorrelations); change MP=M+1 to M=MP−1; remove the initial nested DO loops that compute R from signal samples.

DA2L, shown below, computes Lagrange multipliers. The inputs are MP1, the order plus 1; an array A of inverse filter coefficients; and a gain (excitation power) EIN. The output XL contains MP1 Lagrange multipliers computed by Eq. (7) *except* for the factor of ½ in λ_{h0} = XL(1).

DRC2AU, shown below, computes autocorrelations from reflection coefficients and the square root of the gain. DRC2AU uses a subroutine DRC2E for computing the total power E from reflection coefficients and the square root of the gain.

DA2RC computes reflection coefficients from inverse filter coefficients. A call has the form CALL DA2RC(A,RC,M,IERR), where A and RC are double-precision vectors, and M and IERR are integers. The inputs are the order M and M+1 inverse filter coefficients in A. The outputs are M reflection coefficients in RC and an error indication in IERR; a nonzero value for IERR indicates that a computed magnitude for a reflection coefficient was not less than 1. The subroutine body is adapted from Markel and Gray's subroutine STEPDN [D2, p. 96] with the following changes: convert to double precision and redimension local arrays as for DLEVIN, and return the error indication through IERR instead of printing a message.

REFERENCES

D1. R.W. Johnson and J.E. Shore, "Multisignal Minimum-Cross-Entropy Spectrum Analysis with Weighted Priors," NRL Report in publication; submitted to *IEEE Trans. Acoust., Speech, Signal Process.*

D2. J.D. Markel and A.H. Gray, Jr., *Linear Prediction of Speech*, Springer-Verlag, New York, 1976.

```
      SUBROUTINE MCLPC(RIN,R,W,MDP1,NS,MPR,A,RC,E,L,JAC,IERR)
C...
C...  Multiple signal minimum cross-entropy spectral analysis
C...  with LPC priors
C...
C...  J. Buck
C...  Based on APL program by R. Johnson
C...
C...  Double precision version
C...  Includes degree of belief parameters
C...  Allows different model orders for priors, input
C...
C...  RIN     - Input autocorrelations for total signal
C...  R       - Set of prior autocorrelations - destroyed and replaced
C...            by posterior autocorrelations
C...  W       - Relative weights for each prior estimate (degree of belief)
C...  MDP1    - Total number of input autocorrelations (MD+1)
C...  NS      - Number of signals
C...  MPR     - LPC model order for each prior (vector, length NS)
C...  A       - Autoregressive coefficients for posterior signals
C...  RC      - Reflection coefficients for posterior signals
C...  E       - LPC error powers for posterior signals
C...  L       - Lagrange multipliers for posterior signals
C...  JAC     - Scratch array - MDP1 by MDP1
C...  IERR    - error flag - nonzero if input autos bad
C
      INTEGER K1(30),K2(30),MPR(2)
      REAL*8 R0(30,2),R(30,2),DR(30,2),A(30,2),L(30,2),L0(30,2)
      REAL*8 RC(29,2),W(2)
      REAL*8 B(30),DB(30),RD(30),DL(30),RTOT(30),RIN(30)
      REAL*8 JAC(MDP1,MDP1),E(2),SS(60),ASQ(60),RC2(60)
      REAL*8 S(60,2),SCLU,SCLD,D,DIFF
      LOGICAL SCALED
C
C Initialization
C
      IERR=0
      SCLU=RIN(1)
      SCLD=1./SCLU
C
C Get maximum model order M
C
      MD=MDP1-1
      M=MD
      DO N=1,NS
        M=MAX(MD,MPR(N))
      ENDDO
      MP1=M+1
C
      DO I=1,MP1
          RTOT(I)=RIN(I)*SCLD
          DO N=1,NS
              R(I,N)=R(I,N)*SCLD
          ENDDO
      ENDDO
```

```
        IPASS=0
C
C Get A coefficients, reflection coefficients, lagrange multipliers
C for each prior.
C
        DO N=1,NS
            CALL DLEVIN(MPR(N)+1,R(1,N),A(1,N),E(N),RC(1,N))
            DO I=1,MPR(N)
                IF(ABS(RC(I,N)).GE.1.0)THEN
                    TYPE *,'ILLEGAL PRIOR AUTOCORRELATIONS'
                    IERR=1
                    RETURN
                ENDIF
            ENDDO
C Extend priors to order M
            IF(MPR(N).LT.M) THEN
                DO I=MPR(N)+1,M
                    RC(I,N)=0.
                    A(I+1,N)=0.
                ENDDO
                CALL DRC2AU(SQRT(E(N)),RC(1,N),M,R(1,N))
            ENDIF
            CALL DA2L(A(1,N),L(1,N),MP1,E(N))
            L(1,N)=0.5*L(1,N)
        ENDDO
C Zero work vectors
        DO I=1,MP1
            B(I)=0.0
            DO N=1,NS
                R0(I,N)=R(I,N)
                L0(I,N)=L(I,N)
            ENDDO
        ENDDO
C
C Main loop
C
20      CONTINUE
        IPASS=IPASS+1
        IF(IPASS.GT.100) THEN
            TYPE *,'Convergence failed after 100 passes'
            RETURN
        ENDIF
C
        DO N=1,NS
            DO I=1,MP1
                ASQ(I)=0.
                IM=2*MP1-I
                ASQ(IM)=0.
                DO K=1,I
                    ASQ(I)=ASQ(I)+A(K,N)*A(I-K+1,N)
                    IF(I.LT.MP1)ASQ(IM)=ASQ(IM)+A(MP1-K+1,N)*A(MP1-I+K,N)
                ENDDO
            ENDDO
            CALL DA2RC(ASQ,RC2,2*M,IERR)
            CALL DRC2AU(E(N),RC2,2*M,S(1,N))
```

```
            DO K=1,MP1
                DL(K)=L0(K,N)+B(K)/W(N)-L(K,N)
            ENDDO
            DO J=1,MP1
                DR(J,N)=0.
                DO K=1,MP1
                    DR(J,N)=DR(J,N)-(S(J+K-1,N)+S(ABS(J-K)+1,N))*DL(K)
                ENDDO
            ENDDO
        ENDDO
        DO I=1,MP1+M
            SS(I)=0.
            DO N=1,NS
                SS(I)=SS(I)-S(I,N)/W(N)
            ENDDO
        ENDDO
        DO J=1,MDP1
            DO K=1,MDP1
                JAC(J,K)=SS(J+K-1)+SS(ABS(J-K)+1)
            ENDDO
        ENDDO
C
C Invert Jacobian in place
C
        CALL MINVD(JAC,MDP1,D,K1,K2)
        SCALED = .FALSE.
        DO I=1,MDP1
            RD(I)=RTOT(I)
            DO N=1,NS
                RD(I)=RD(I)-R(I,N)-DR(I,N)
            ENDDO
        ENDDO
        DO I=1,MDP1
            DB(I)=0.
            DO J=1,MDP1
                DB(I)=DB(I)+RD(J)*JAC(J,I)
            ENDDO
            B(I)=B(I)+DB(I)
        ENDDO
C
C Compute delta-R
C
        DO N=1,NS
            DO J=1,MP1
                DO K=1,MP1
                    DR(J,N)=DR(J,N)-(S(J+K-1,N)+S(ABS(J-K)+1,N))*DB(K)/W(N)
                ENDDO
            ENDDO
        ENDDO
C
C Check constraints
C
        D = 0.0
        DO I=1,MDP1
            DIFF = RTOT(I)
```

```
            DO N=1,NS
                DIFF = DIFF - R(I,N) - DR(I,N)
            ENDDO
            D = MAX(D,ABS(DIFF))
        ENDDO
        IF (D.GE.1.0E-5) THEN
            TYPE 70, IPASS, D
70          FORMAT(' Constraints violated on pass',I4,
     +                          '.  Rel. discrepancy',1PE13.6,'.')
            RETURN
        ENDIF
C
C Save old value of R
C
        DO I=1,MP1
            DO N=1,NS
                R0(I,N)=R(I,N)
            ENDDO
        ENDDO
C
C Calculate new R's, test for legality
C
120     DO I=1,MP1
            DO N=1,NS
                R(I,N)=R0(I,N)+DR(I,N)
            ENDDO
        ENDDO
        DO N=1,NS
            IF(R(1,N).LE.0.0)GOTO 150
            CALL DLEVIN(MP1,R(1,N),A(1,N),E(N),RC(1,N))
            DO I=1,M
                IF(ABS(RC(I,N)).GE.1.0)GOTO 150
            ENDDO
        ENDDO
        GOTO 160
C
C New autocorrelations not feasible; reduce size of jump
C
150     DO I=1,MP1
            DO N=1,NS
                DR(I,N)=0.75*DR(I,N)
            ENDDO
        ENDDO
        SCALED = .TRUE.
        GOTO 120
C
C Get new Lagrange multipliers
C
160     DO N=1,NS
            CALL DA2L(A(1,N),L(1,N),MP1,E(N))
            L(1,N)=0.5*L(1,N)
        ENDDO
C
C Convergence check
C
```

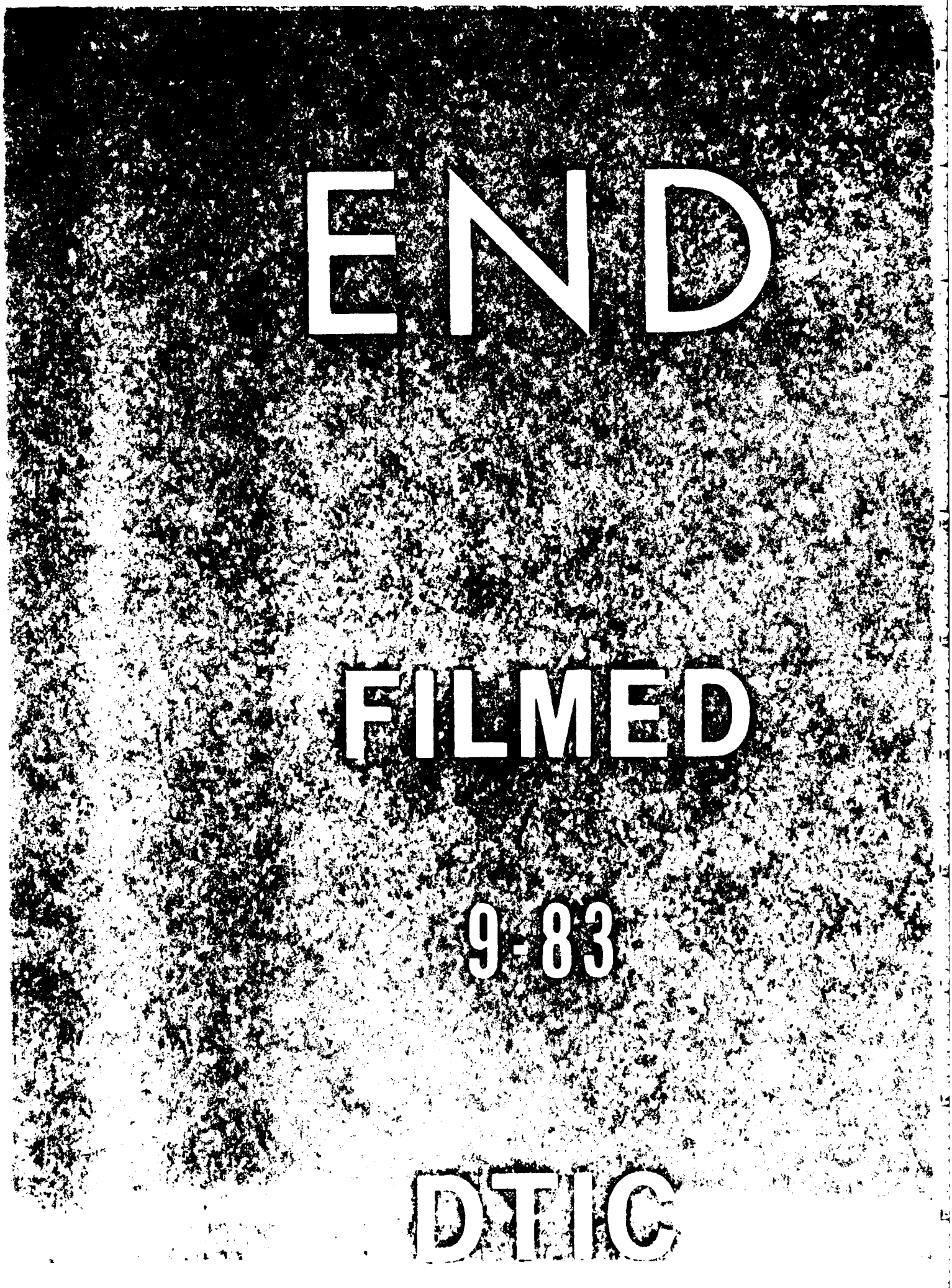
```
IF (SCALED) GOTO 20
DO I=1,MP1
    DO N=1,NS
        IF(R0(I,N).EQ.0.)GOTO 20
        IF(ABS(DR(I,N)/R0(I,N)).GT.1.E-5)GOTO 20
    ENDDO
ENDDO
DO N=1,NS
    E(N)=E(N)*SCLU
    DO I=1,MP1
        R(I,N)=R(I,N)*SCLU
    ENDDO
ENDDO
IERR=0
RETURN
END
```

```
      SUBROUTINE DA2L(A,XL,MP1,EIN)
      REAL*8 A(MP1),XL(MP1),EIN,EO
      EO=1./EIN
      DO 10 J=1,MP1
        XL(J)=0.0
        DO 5 I=1,MP1-J+1
          XL(J)=XL(J)+A(I)*A(I+J-1)
5       CONTINUE
        XL(J)=XL(J)*EO
10    CONTINUE
      RETURN
      END
```

```
      SUBROUTINE DRC2AU(RMSS,RC,N,R)
C...
C...  CONVERT GAIN AND REFLECTION COEFFICIENTS TO AUTOCORRELATIONS
C...  DOUBLE PRECISION VERSION
C...
C...  INPUTS:
C...
C...  RMSS - SQRT OF LPC GAIN
C...  RC   - REFLECTION COEFFICIENT ARRAY
C...  N    - NUMBER OF REFLECTION COEFFICIENTS
C...
C...  OUTPUTS:
C...
C...  R  - ARRAY OF N+1 AUTOCORRELATIONS
C...
      REAL*8 RC(1),R(1),A(60),A1(60),RMSS,E
      CALL DRC2E(RC,N,RMSS,R(1))
      E=R(1)
      DO 50 M=1,N
        R(M+1)=-E*RC(M)
        E=E*(1-RC(M)**2)
        A1(M)=-RC(M)
        IF(M.EQ.1)GOTO 20
        DO 10 J=1,M-1
          R(M+1)=R(M+1)+A(J)*R(M-J+1)
10        A1(J)=A(J)+RC(M)*A(M-J)
20      DO 40 J=1,M
40        A(J)=A1(J)
50    CONTINUE
      RETURN
      END



      SUBROUTINE DRC2E(RC,M,RMSS,E)
      REAL*8 RC(30),RMSS,E,CF
      CF=1.
      IF(M.EQ.0)GOTO 20
      DO 10 I=1,M
10    CF=CF*(1.-RC(I)**2)
20    E=RMSS**2/CF
      RETURN
      END
```


END
FILMED
9-83
DTIC